Programmation Python

De la base aux concepts avancés de Python

Audrey Kouessi FANGNON & Johannès HOUNSINOU

Bénin Excellence - Fondation Vallet / FSS COTONOU

4 janvier 2025

Plan du cours

1. Chapitre 4 : Les séquences de données.
   - Les chaînes de caractères
   - Les Listes et les Tuples

Introduction

Qu'est-ce que c'est ?

Les séquences en Python sont des structures de données composées d'entités plus petites. Il s'agit entre autres :

- des chaines de caractere (string) ;
- des listes (list) ;
- des tuples ;
- des dictionnaires (dict) et
- des ensembles (set)

A quoi servent-ils ?

Chaque types de séquences de données à ses propres caractéristiques et offre des avantages pour organiser et manipuler l'information. Ce qui ouvre ainsi la porte à des solutions élégantes et efficaces.

Manipulation des Strings

Indexing (Indexage)

Les chaînes de caractères (ou strings) représente des séquences de caractères Unicode accessibles par un ***indice*** (ou ***indexe***).

- Le premier caractère (à gauche) étant à la position ***0***.
- Les indices peuvent être négatif, on compte alors de ***-1*** en partant de la ***droite***

Slicing (ou tranche)

Le ***slicing*** permet de récupérer une sous-chaîne d'une chaîne. La syntaxe est : `ma_chaine[start:stop:step]`.
Par défaut : `start=0` , `stop=len(ma_chaine)` et `step=1`.

Le début (`start`) est toujours inclus et la fin (`stop`) est exclus ; cela préserve : `ma_chaine[:i] + ma_chaine[i:] = ma_chaine`

Manipulation des Strings - Indexage et Tranche

Exemples :

```
>>> chaine = "Bénin Excellence!"
>>> chaine[0]      # premier caractère de chaine
>>> chaine[3]      # quatrième caractère
>>> chaine[-1]      # dernier caractère
>>> chaine[-2]      # avant dernier caractère
>>> chaine[6:11]      # tranche du septième au douzième
    caractère
>>> chaine[6:]      # tranche du septième au dernier
    caractère
>>> chaine[:5]      # tranche du premier au sixième
    caractère
>>> chaine[::-1]      # tranche du premier au dernier
    caractère dans le sens inverse
```

Manipulation des Strings - La Casse

La casse

En Python, les Strings possède plusieurs méthodes permettant de modifier la ***casse***. Il s'agit entre autre de : `str.lower()`, `str.upper()`, `str.title()`, `str.capitalize()`, `str.swapcase()`...

Exemple :

```
>>> texte = "le Bénin est magnifique."
>>> texte.lower()      # Tout en miniscule
>>> texte.upper()      # Tout en majuscule
>>> texte.title()      # Majuscule à chaque mot
>>> texte.capitalize()      # Majuscule au début de
    phrase.
>>> texte.swapcase()      # inverse la casse
```

Manipulation des Strings - Autre méthodes utiles

Les occurrences

Pour compter le nombre d'apparition (ou occurrence) d'une sous-chaine dans une chaîne, on utilise la méthode : `str.count(sousChaine)`

Rechercher et Remplacer

La méthode `str.replace(old, new)` permet de remplacer toutes les occurences de la sous-chaine `old` par la chaine `new`.

Exemple

```
>>> texte = "Bénin est magnifique."
>>> texte.count("nin")      # compte les apparition de
    "nin"
>>> texte.replace("nin", "linda")      # remplacement
```

Coding Time - Strings

Problème - Palindrome

Écrivez un programme qui fait appel à une fonction booléen palindrom(phrase) permettant de vérifier si un mot ou une phrase (non accentué mais pouvant contenir des ponctuations) saisie par l'utilisateur est un *Palindrome*.
Exemple : Radar; Ressasser; ete; Zeus a ete a Suez.; A l'etape, epate-la!; ...

Recommandation

- `dir(str)  # renvoie tous les attributs et méthodes de l'objet str`
- `help(str)  # affiche l'aide complète sur toutes les méthodes de l'objet str`

Premier pas avec les listes

Liste en Python

Les listes représentent l'une des structures de données les plus polyvalentes et flexible en Python. Elle permettent de stocker une collection ***ordonnée*** d'éléments ***modifiable*** à volonté qui peuvent être de différents types.

Création

Pour créer uns liste, on utilise des crochets [ ]
Exemple :

```
>>> ma_liste = [47, "Bénin", 3, [1,2], 7]
```

- `len(ma_liste)` : permet de connaitre le nombre d'éléments dans la liste `ma_liste`
- Le slicing et l'indexage fonctionne de la même manière sur les liste que sur les strings.

Manipulation des listes

Exemple

```
>>> ma_liste = ["X", 5, "A", 47, "X", 9, "+"]
>>> len(ma_liste)      # Longueur de la liste
>>> ma_liste[::2]      # Tranche des éléments d'indice
    pair
>>> ma_liste[1::2]      # Tranche des éléments d'indice
    impair
>>> ma_liste[1] = "FSS"      # modification de
    l'élément d'indice 1
>>> ma_liste      # afficher ma_liste
```

Manipulation des listes

Manipulation

Le type liste dispose de plusieurs méthodes supplémentaires pour faciliter les manipulation.

- `list.append(x)` : ajoute l'élément `x` à la fin de la liste.
- `list.extend(iterable)` : étend la liste en y ajoutant toue les éléments de `l'itérable`
- `list.insert(i, x)` : insert l'élément x à la position i.
- `list.remove(x)` : supprime de la liste le premier élément dont la valeur égale `x`.
- `list.pop(index=-1)` : supprime et retourne l'élément à la position `index`

Manipulation des listes

Manipulation

- `list.clear()` : supprime tous les élément de la liste.
- `list.index(x, start=0, stop=9223372036854775807)` : renvoie la position du premier élément égale à x de `start` (inclus) à `end (exlus)`.
- `list.count(x)` : renvoie le nombre d'éléments ayant la valeur `x` dans la liste.
- `list.sort(*, key=None, reverse=False)` : permet de ranger la liste suivant l'ordre croissant par défaut et retourne `None`).
- `list.reverse()` : renverse l'ordre des éléments dans une liste.

Manipulation des listes

Exemple 1 :

```
>>> liste1 = ["Sagesse", "Pouvoir","limité"]
>>> liste1.pop()      # suppression du dernier élément
>>> liste1.append("Lumière")      # ajout d'élément à
    la fin
>>> liste1[1:1] = ["Eternité"]      # ajout d'élément à
    la position 1
>>> chaine = " et ".join(liste1)      # Assembler une
    liste en chaine
>>>
>>> chaine = "ordinateur;serveur;internet;réseaux"
>>> liste2 = chaine.split(";")      # Diviser chaîne en
    liste
>>> liste2      # Affichage
>>>
```

Shallow copy - Deep copy

Code à tester:

```
>>> fruits = ["orange","banane","ananas"]
>>> copie_fruit = fruits     # copie superficielle
>>> copie_fruit.remove("banane")     # supprime
    "banane" de copie_fruit
>>> copie_fruit     # affiche copie_fruit
>>> fruit     # Attention, la liste fruit est modifiée
    aussi
```

Que s'est-il passé ?

Lors de la ligne 2, Python n'effectue pas de copie mais crée plutôt un ***alias***. Pour effectuer une véritable copie appélée ***copie profonde***, il faudra utiliser la fonction deepcopy du module copy ; ou simplement faire une copie des élément un à un.

Shallow copy - Deep copy

Exemple 1 : utilisation de la fonction deepcopy

```
>>> from copy import deepcopy      #importation de
    deepcopy
>>> fruits = ["orange","banane","ananas"]
>>> copie_fruit = deepcopy(fruits)      # copie
    profonde
>>> copie_fruit.remove("banane")      # supprime
    "banane" de copie_fruit
>>> copie_fruit      # affiche copie_fruit
>>> fruits      # La liste fruits n'est pas modifiée
```

Shallow copy - Deep copy

Exemple 2 : copie des éléments un à un

```
>>> fruits = ["orange","banane","ananas"]
>>> copie_fruit = []
>>> for element in fruits:
...     copie_fruit.append(element) # ajout de element
>>> copie_fruit # affiche copie_fruit
>>> fruits     # La liste fruit n'est pas modifiée
```

Les listes en compréhension

Création rapide de listes

Les listes en compréhension sont une syntaxe en Python qui permet de créer des listes à partir d'une expression unique souvent combinée avec des itérations et des filtres. La syntaxe est :
`ma_liste = [expresion for element in sequence if condition]`

Exemples :

```
>>> liste = [x for x in range(10)]   # Liste des
    nombres de 0 à 9
>>> carre = [x*x for x in range(10)]   # Liste des
    carrés des nombres de 0 à 9
>>> liste = [x for x in range(10) if x%2 == 0]   #
    Liste des nombres pairs compris entre 0 à 9
```

Premier pas avec les Tuples

Tuple en Python

Les tuples sont un ensemble ordonné d'éléments de tout types comme les listes. La différence entre un Tuple et une liste est qu'une liste est modifiable (mutable) et un Tuple, non. On dit que les tuples sont immutables.

Création

Pour créer un tuple, on utilise des parenthèses ()
Exemple :

```
>>> mon_tuple = (47, "Bénin", 3, [1,2], 7)
```

Slicing - Indexing

Les tranches et les indexages fonctionnent de la même manière chez les tuples que chez les listes.

Les tuples en compréhension ?

Création rapide de tuples

Il n'existe pas de tuple en compréhension dans le langage Python mais nous pouvons simuler cela grace à la fonction tuple.

Exemples :

```
>>> tup1 = tuple([x for x in range(10)])   #
    conversion de liste en tuple
>>> tup1
>>> carre = tuple( (x*x for x in range(10)) )   #
    Tuple à partir d'un générateur
>>> carre
```

Bonus sur les Listes et Tuples

enumerate()

La fonction `enumerate(liste)` retourne ***un à un*** un tuple contenant l'indice et l'élément se trouvant à ce indice de `liste`

Exemple :

```
>>> liste = ['L', 'M', 'N', 'O', 'P']
>>> for index,element in enumerate(liste):
...     print(f"Indice {index} ==> {element}")
```

zip()

La fonction `zip(*listes)` permet de parcourir plusieurs listes en même temps.

Exemple :

```
>>> a = [1,2,3]; b = [4,5,6]; c = [7,8,9]
>>> for item in list(zip(a,b,c))
...     item
```

Coding Time - Liste et Tuples

Problème A : il était une fois... SommeProduitMoyenne - Épisode_1

Ecrivez un programme qui appelle une fonction caracteristique(liste) qui prend en argument une liste et renvoie un tuple (somme, produit, moyenne).
Pour une liste L = [2, 5, 9] le programme affiche :

```
somme = 16
produit = 90
moyenne = 5,3333333
```

Problème B : SommeProduitMoyenne - Épisode_2

Ecrivez un programme qui complète l'Épisode_1 et permet au programme de récupérer désormais la liste en demandant à l'utilisateur de saisir les éléments sans espace en les séparant par un tiret.

Coding Time - Liste et Tuples

Problème C - Loterie Gagnante

Ecrivez un programme de loterie qui demande à l'utilisateur de choisir 5 numéros tous différents entre 1 et 50 inclus et qui effectue ensuite un tirage aléatoire sur les mêmes critères puis indique le nombre de numéros gagnants.

Exemple :

```
Entrez le 1e numéro entre 1 et 50 inclus : 4
Entrez le 2e numéro entre 1 et 50 inclus : 50
Entrez le 3e numéro entre 1 et 50 inclus : 25
Entrez le 4e numéro entre 1 et 50 inclus : 17
Entrez le 5e numéro entre 1 et 50 inclus : 47
Votre choix: 4 - 50 - 25 - 17 - 47
Le tirage: 1 - 28 - 47 - 19 - 24
Vous avez 1 numéro gagnant(s).
```

Ensembles et dictionnaires

Les séquences de données non ordonnées

En Python, les ***ensembles*** (‘set‘) et les ***dictionnaire*** (‘dict‘) représentent principalement des structures qui stockent des données sans imposer un ordre particulier.

Les ensembles

Un ***ensemble*** est une collection non ordonnée d’***éléments uniques***. Il est défini en utilisant la syntaxe ***‘{ }‘*** ou la fonction ***‘set()‘***.

Les dictionnaires

Un ***dictionnaire*** est une collection non ordonnée de paires ***clé-valeur***. Il est défini en utilisant la syntaxe ***‘{clé :valeur}‘*** ou la fonction ***‘dict()‘***.

Ensembles et dictionnaires

Exemples :

```
>>> ensemble1 = {"Bénin", 1960, 1, 8, "indépendance"}
    # Création d'ensemble.
>>> liste_impairs = [x for x in range(10) if x%2 != 0]
>>> ensemble2 = set(liste_impairs)     # Création
    ensemble à partir de liste
>>>
>>> dictionnaire = {"Nom":"MUSK", "Prenoms":"Elon",
    "Naissance": [28,06,1971], "Entreprise": {"Tesla",
    "SpaceX", "x.com", "Neuralink"},
    "Fortune($)":251.3e9 }
>>>
```

Ensembles et dictionnaires

Utilités

- Les ***ensembles*** sont idéales pour tester l'appartenance d'un élément (en utilisant ***'in'***) et éliminer les doublons.
- Les ***dictionnaires*** permettent d'associer des informations (appelées valeurs) à des clés puis d'avoir un accès rapide à ces informations via les clés.
- Les ensembles et dictionnaires disposent de plusieurs méthodes pour faciliter la gestion des données.

Opérations courantes sur les ensembles.

```
>>> ensemble1 = {"Bénin", 1960, 1, 8, "colonisation"}
>>> ensemble1.discard("colonisation")   # Suppression
    élément.
>>> "indépendance" in ensemble1
>>> ensemble.add("indépendance")     # Ajout élément.
>>> "indépendance" in ensemble1
>>> ensemble2 = {1, 3, 5, 8 }
>>> ensemble1.union(ensemble2)     # Reunion
>>> ensemble1.intersection(ensemble2)   # Intersection
>>> ensemble1.difference(ensemble2)     # Différence
>>> help(ensemble1)     # Aide complète de l'objet
```

Opérations courantes sur les dictionnaires.

```
>>> fournitures = {"cahiers":134,
    "stylos":{"rouge":41,"bleu":74}, "gommes": 85}
>>> fournitures.values()   # Retourne les valeurs.
>>> fournitures.keys()   # Retourne les clés.
>>>
>>> for cle in fournitures.keys():
...     print(cle, " Nombre = ", fournitures[cle])
>>>
>>> fournitures["agrafes"] = 47 }     # Ajout élément
>>> fournitures["stylos"]["noir"] = 23 }
>>> fournitures.pop("gommes")   # suppression élément
>>> help(fournitures)     # Aide complète de l'objet
```

Ensemble et dictionnaire en compréhension

Création rapide d'ensemble et de dictionnaire

Comme dans le cas des listes, Python nous offre la possibilité de créer des ensembles en compréhension et des dictionnaires en compréhension.

Dictionnaire en compréhension

```
mon_dic = { cle:valeur for cle, valeur in iterable if condition}
```

Ensemble en compréhension

```
mon_ensemble = { expression for element in iterable if condition}
```

Ensemble et dictionnaire en compréhension

Exemples :

```
>>> ens = {x for x in range(10) }   #
>>> ens
>>> names = ["John", "Melquis", "Steeve", "Toto"]
>>> marks = [20, 15, 19, 4]
>>> dic = {name:mark for name, mark in zip(names,
    marks) if mark >= 10}   # dictionnaire en
    compréhension
>>> dic
```

Coding Time - Les séquences de données.

Problème D : Gestion de Banques

Écrivez un programme simulant le fonctionnement d'une banque en stockant le solde des comptes dans un dictionnaire. Il devra permettre le dépôt et le retrait de sommes d'argent.
Voici la liste (Clients ==> Solde) de la banque:

- Paul ===> 400 000 FCFA
- Marie ===> 550 200 FCFA
- Jean ===> 90 000 FCFA
- Pauline ===> 1 000 500 FCFA

Consignes

A chaque itération, le programme affiche la liste des comptes clients avec leurs solde respectifs puis le menu comme suit :

Coding Time - Les séquences de données.

```
 Banque

Client      Solde(FCFA)
----------------------
Marie       550200
Pauline     1000500
Jean        90000
Paul        400000

Choisissez une opération (D : Dépot - R : Retrait - Q
: Quitter) : d
Entrez le nom du compte du client sur lequel effectuer
le dépot : Jean
Quel montant voulez-vous déposez ? 200000
Opération effectuée.
```

Coding Time - Les séquences de données.

```
 Banque

Client      Solde(FCFA)
----------------------
Marie       550200
Pauline     1000500
Jean        290000
Paul        400000

Choisissez une opération (D : Dépot - R : Retrait - Q
: Quitter) : r
Entrez le nom du client sur lequel effectuer le
retrait : Paul
Quel montant voulez-vous retirer ? 500000
Opération impossible : votre solde est insuffisant.
```